Story 1.3: Kafka Infrastructure & Schema Registry

Status: ready-for-dev

Story

As a Developer,
I want Kafka cluster con Schema Registry configurado para eventos Avro,
so that Puedo publicar domain events con garantía de schema versionado y backward compatibility.

Acceptance Criteria

AC1: Kafka + Zookeeper + Schema Registry Docker Compose

Given El proyecto tiene Docker Compose configurado
When Agrego servicios de Kafka al `docker-compose.yml`
Then

- Servicio `zookeeper` configurado:
  - Imagen: `confluentinc/cp-zookeeper:7.5.0`
  - Puerto: `2181:2181`
  - Configuración: `ZOOKEEPER_CLIENT_PORT=2181`
- Servicio `kafka` configurado:
  - Imagen: `confluentinc/cp-kafka:7.5.0`
  - Puerto: `9092:9092` (external), `29092:29092` (internal)
  - Bootstrap servers: `localhost:9092` (dev), `kafka:29092` (container-to-container)
  - Depende de `zookeeper`
- Servicio `schema-registry` configurado:
  - Imagen: `confluentinc/cp-schema-registry:7.5.0`
  - Puerto: `8081:8081`
  - URL: `http://localhost:8081`
  - Depende de `kafka`
- Comando `docker-compose up -d` levanta los 3 servicios exitosamente
- Healthchecks configurados para Kafka y Schema Registry

AC2: Spring Kafka Dependencies

Given El proyecto tiene Spring Boot 3.2+
When Agrego dependencias de Kafka
Then

- `pom.xml` incluye:
  - `spring-kafka` (versión gestionada por Spring Boot)
  - `kafka-avro-serializer` (Confluent 7.5.0)
  - `avro` (Apache Avro 1.11+)
  - `kafka-streams-test-utils` (test scope)
  - `spring-kafka-test` (test scope)

AC3: Kafka Configuration (Spring Boot)

Given Kafka running en Docker
When Configuro `application-local.yml`
Then

- Producer configuration:
  - `spring.kafka.bootstrap-servers=localhost:9092`
  - `spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer`
  - `spring.kafka.producer.value-serializer=io.confluent.kafka.serializers.KafkaAvroSerializer`
  - `spring.kafka.producer.acks=all` (strong durability)
  - `spring.kafka.producer.compression-type=snappy`
  - `spring.kafka.producer.max-in-flight-requests-per-connection=5`
- Schema Registry configuration:
  - `spring.kafka.properties.schema.registry.url=http://localhost:8081`
- Admin configuration:
  - `spring.kafka.admin.auto-create=true`

AC4: Avro Schema Definition

Given Schema Registry configurado
When Defino esquema Avro para eventos de dominio
Then

- Archivo `src/main/resources/kafka/schemas/signature-event.avsc` creado
- Esquema define:
  - `namespace`: `com.bank.signature.event`
  - `name`: `SignatureEvent`
  - `type`: `record`
  - Campos comunes: `eventId`, `eventType`, `aggregateId`, `aggregateType`, `timestamp`, `traceId`
  - Campo `payload`: Union type con 8 event types:
    - `SIGNATURE_REQUEST_CREATED`
    - `CHALLENGE_SENT`
    - `CHALLENGE_COMPLETED`
    - `CHALLENGE_FAILED`
    - `SIGNATURE_COMPLETED`
    - `SIGNATURE_FAILED`
    - `FALLBACK_TRIGGERED`
    - `PROVIDER_DEGRADED`
  - Backward compatibility validada

AC5: Kafka Topic Creation

Given Kafka Admin configurado con `auto-create=true`
When La aplicación inicia
Then

- Topic `signature.events` creado automáticamente:
  - Partitions: 12 (para throughput)
  - Replication factor: 1 (dev), 3 (prod)
  - Retention: 7 días
  - Compression: `snappy`
- Topic `signature.events.dlq` (Dead Letter Queue) creado:
  - Partitions: 3
  - Replication factor: 1 (dev), 3 (prod)
  - Retention: 30 días

AC6: KafkaTemplate Configuration

Given Spring Kafka configurado
When Creo `KafkaConfig.java`
Then

- Bean `KafkaTemplate<String, GenericRecord>` configurado
- ProducerFactory con:
    - Key serializer: `StringSerializer`
    - Value serializer: `KafkaAvroSerializer`
    - Idempotence habilitado (`enable.idempotence=true`)
    - Transactional ID configurado (para exactly-once semantics en futuro)
- Default topic: `signature.events`

AC7: Schema Registration in Schema Registry

Given Schema Registry running
When Registro esquema Avro
Then

- Esquema `signature-event-value` registrado en Schema Registry
- Subject: `signature.events-value` (key strategy: TopicNameStrategy)
- Compatibility mode: `BACKWARD` (permite agregar campos opcionales)
- Schema ID asignado (e.g., 1)
- GET `http://localhost:8081/subjects` retorna `["signature.events-value"]`
- GET `http://localhost:8081/subjects/signature.events-value/versions/latest` retorna schema

AC8: Kafka Health Check

Given Kafka configurado en Spring Boot
When Configuro Actuator health check
Then

- Endpoint `/actuator/health/kafka` retorna `{"status":"UP","details":{"kafkaConsumers":"UP","kafkaProducers":"UP"}}`
- Health check verifica:
    - Conexión a Kafka broker exitosa
    - Producer está listo para enviar mensajes

- Si Kafka está down, endpoint retorna `{"status":"DOWN"}`

AC9: Maven Avro Plugin Configuration

Given Esquema Avro definido en `.avsc`
When Configuro `avro-maven-plugin` en `pom.xml`
Then

- Plugin `avro-maven-plugin` configurado en `<build><plugins>`
- Goal: `schema` (genera clases Java desde `.avsc`)
- Source directory: `src/main/resources/kafka/schemas`
- Output directory: `target/generated-sources/avro`
- Comando `./mvnw clean compile` genera clases:
  - `com.bank.signature.event.SignatureEvent`
  - Builders, getters, setters para cada event type
- Clases generadas disponibles en classpath

AC10: Integration Test with Embedded Kafka

Given Spring Kafka Test configurado
When Creo test de integración `KafkaInfrastructureIntegrationTest.java`
Then

- Test usa `@EmbeddedKafka` con:
  - Topics: `signature.events`, `signature.events.dlq`
  - Partitions: 3 (embedded)
  - Broker properties: auto-create topics, port 9093
- Test verifica:
  - KafkaTemplate puede enviar mensaje GenericRecord
  - Mensaje se serializa correctamente con Avro
  - Schema Registry (mock) valida esquema
  - Mensaje llega a topic `signature.events`
- Test pasa en `mvn verify`

AC11: Kafka Configuration Profiles

Given Múltiples entornos (local, uat, prod)
When Configuro profiles en `application-{profile}.yml`
Then

- `application-local.yml`:
  - `bootstrap-servers: localhost:9092`
  - `schema.registry.url: http://localhost:8081`
- `application-uat.yml` (futuro):
  - `bootstrap-servers: kafka-uat.internal:9092`
  - `schema.registry.url: http://schema-registry-uat.internal:8081`
- `application-prod.yml` (futuro):
  - `bootstrap-servers: kafka-prod.internal:9092`
  - `schema.registry.url: http://schema-registry-prod.internal:8081`
  - `producer.acks: all`
  - `producer.enable.idempotence: true`

AC12: Documentation & README Update

Given Kafka infrastructure configurado
When Actualizo documentación
Then

- `README.md` actualizado con sección "Kafka Setup":
  - Comandos Docker Compose para Kafka
  - Comandos para verificar topics: `docker exec kafka kafka-topics --bootstrap-server localhost:9092 --list`
  - Comandos para Schema Registry: `curl http://localhost:8081/subjects`
- `docs/development/kafka-messaging.md` creado con:
  - Avro schema evolution guidelines
  - Event publishing patterns
  - Testing strategy con Embedded Kafka
  - Troubleshooting (Kafka connection errors, schema validation failures)
- `CHANGELOG.md` actualizado con Story 1.3 entry

Tasks / Subtasks

Task 1: Add Kafka Services to Docker Compose (AC: #1)

- ◯ 1.1. Agregar servicio `zookeeper` a `docker-compose.yml`:
  - Imagen: `confluentinc/cp-zookeeper:7.5.0`
  - Puerto: 2181
  - Variable: `ZOOKEEPER_CLIENT_PORT=2181`
- ◯ 1.2. Agregar servicio `kafka` a `docker-compose.yml`:
  - Imagen: `confluentinc/cp-kafka:7.5.0`
  - Puertos: 9092 (external), 29092 (internal)
  - Variables: `KAFKA_BROKER_ID`, `KAFKA_ZOOKEEPER_CONNECT`, `KAFKA_ADVERTISED_LISTENERS`, `KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR`
  - Depende de: `zookeeper`
- ◯ 1.3. Agregar servicio `schema-registry` a `docker-compose.yml`:
  - Imagen: `confluentinc/cp-schema-registry:7.5.0`
  - Puerto: 8081
  - Variables: `SCHEMA_REGISTRY_HOST_NAME`, `SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS`
  - Depende de: `kafka`
- ◯ 1.4. Agregar healthchecks para Kafka y Schema Registry
- ◯ 1.5. Verificar: `docker-compose up -d` levanta los 3 servicios

Task 2: Add Spring Kafka Dependencies to pom.xml (AC: #2)

- ◯ 2.1. Agregar `spring-kafka` (Spring Boot managed version)
- ◯ 2.2. Agregar `kafka-avro-serializer` (Confluent 7.5.0)
- ◯ 2.3. Agregar `avro` (Apache Avro 1.11+)
- ◯ 2.4. Agregar `kafka-streams-test-utils` (test scope)
- ◯ 2.5. Agregar `spring-kafka-test` (test scope)

Task 3: Configure Kafka in application-local.yml (AC: #3)

- ◯ 3.1. Configurar `spring.kafka.bootstrap-servers=localhost:9092`
- ◯ 3.2. Configurar producer serializers (String + KafkaAvroSerializer)
- ◯ 3.3. Configurar producer properties: `acks=all`, `compression-type=snappy`, `max-in-`

`flight=5`

- ◯ 3.4. Configurar Schema Registry URL: `http://localhost:8081`
- ◯ 3.5. Configurar admin auto-create: `spring.kafka.admin.auto-create=true`

Task 4: Define Avro Schema for Domain Events (AC: #4)

- ◯ 4.1. Crear directorio `src/main/resources/kafka/schemas/`
- ◯ 4.2. Crear archivo `signature-event.avsc` con namespace `com.bank.signature.event`
- ◯ 4.3. Definir campos comunes: `eventId`, `eventType`, `aggregateId`, `timestamp`, `traceId`
- ◯ 4.4. Definir union type `payload` con 8 event types (SIGNATURE_REQUEST_CREATED, CHALLENGE_SENT, etc.)
- ◯ 4.5. Validar esquema con Avro tools: `java -jar avro-tools.jar compile schema signature-event.avsc .`

Task 5: Configure Kafka Topics (AC: #5)

- ◯ 5.1. Crear `KafkaTopicConfig.java` con beans `NewTopic`
- ◯ 5.2. Configurar topic `signature.events`:
  - 12 partitions, replication factor 1 (dev)
  - Retention 7 días, compression snappy
- ◯ 5.3. Configurar topic `signature.events.dlq`:
  - 3 partitions, replication factor 1 (dev)
  - Retention 30 días
- ◯ 5.4. Verificar topics con: `docker exec kafka kafka-topics --bootstrap-server localhost:9092 --list`

Task 6: Configure KafkaTemplate Bean (AC: #6)

- ◯ 6.1. Crear `KafkaConfig.java` en `infrastructure/config/`
- ◯ 6.2. Configurar `ProducerFactory<String, GenericRecord>` con KafkaAvroSerializer
- ◯ 6.3. Configurar `KafkaTemplate<String, GenericRecord>` bean
- ◯ 6.4. Habilitar idempotence: `enable.idempotence=true`
- ◯ 6.5. Configurar default topic: `signature.events`

Task 7: Register Schema in Schema Registry (AC: #7)

- ◯ 7.1. Iniciar Schema Registry: `docker-compose up -d schema-registry`
- ◯ 7.2. Configurar compatibility mode: `BACKWARD`
- ◯ 7.3. Registrar schema vía API o auto-registro en primer envío
- ◯ 7.4. Verificar: `curl http://localhost:8081/subjects`
- ◯ 7.5. Verificar schema: `curl http://localhost:8081/subjects/signature.events-value/versions/latest`

Task 8: Configure Kafka Health Check (AC: #8)

- ◯ 8.1. Verificar `spring-boot-starter-actuator` está en `pom.xml`
- ◯ 8.2. Configurar `management.health.kafka.enabled=true` en `application.yml`
- ◯ 8.3. Exponer endpoint en `management.endpoints.web.exposure.include` (ya configurado)
- ◯ 8.4. Verificar: `curl http://localhost:8080/actuator/health/kafka`
- ◯ 8.5. Test: detener Kafka, verificar health check retorna DOWN

Task 9: Configure Maven Avro Plugin (AC: #9)

- ◯ 9.1. Agregar `avro-maven-plugin` a `pom.xml` en `<build><plugins>`
- ◯ 9.2. Configurar goal `schema` para generar clases Java
- ◯ 9.3. Configurar source directory: `src/main/resources/kafka/schemas`
- ◯ 9.4. Configurar output directory: `target/generated-sources/avro`
- ◯ 9.5. Ejecutar: `./mvnw clean compile`
- ◯ 9.6. Verificar clases generadas: `com.bank.signature.event.SignatureEvent`

Task 10: Create Integration Test with Embedded Kafka (AC: #10)

- ◯ 10.1. Crear `KafkaInfrastructureIntegrationTest.java` en `src/test/java/.../infrastructure/`
- ◯ 10.2. Configurar `@EmbeddedKafka` con topics: `signature.events`, `signature.events.dlq`
- ◯ 10.3. Autowire `KafkaTemplate<String, GenericRecord>`
- ◯ 10.4. Test method: `testKafkaTemplateSendsAvroMessage()`
  - Crear GenericRecord con evento SIGNATURE_REQUEST_CREATED
  - Enviar con KafkaTemplate

    - Consumir mensaje con KafkaConsumer
    - Verificar serialización Avro correcta
- ◯ 10.5. Ejecutar: `./mvnw verify`

Task 11: Configure Kafka Profiles for Multiple Environments (AC: #11)

- ◯ 11.1. Configurar `application-local.yml` con bootstrap-servers localhost
- ◯ 11.2. Crear `application-uat.yml` con Kafka internal URLs (placeholder)
- ◯ 11.3. Crear `application-prod.yml` con Kafka internal URLs + idempotence (placeholder)
- ◯ 11.4. Documentar diferencias en `docs/development/kafka-messaging.md`

Task 12: Update Documentation (AC: #12)

- ◯ 12.1. Actualizar `README.md` con sección "Kafka Setup":
    - Comandos Docker Compose
    - Comandos para listar topics
    - Comandos Schema Registry (curl)
- ◯ 12.2. Crear `docs/development/kafka-messaging.md`:
    - Avro schema evolution guidelines
    - Event publishing patterns
    - Testing strategy
    - Troubleshooting
- ◯ 12.3. Actualizar `CHANGELOG.md` con Story 1.3 entry
- ◯ 12.4. Agregar comentarios en `KafkaConfig.java` explicando configuraciones

Dev Notes

Architecture Patterns & Constraints

- **Event-Driven Architecture**: Kafka como backbone para eventos de dominio (Outbox pattern en Story 1.2)
- **Avro Serialization**: Esquema versionado con backward compatibility garantiza evolución segura
- **Schema Registry**: Confluent Schema Registry valida esquemas antes de publicar (fail fast)
- **Idempotent Producer**: `enable.idempotence=true` garantiza exactly-once delivery en caso de retries
- **Partitioning Strategy**: 12 partitions para `signature.events` permite throughput alto (parallel consumers)

Source Tree Components to Touch

```
signature-router/
├── pom.xml                                       # [MODIFY] agregar spring-kafka,
kafka-avro-serializer, avro, avro-maven-plugin
├── docker-compose.yml                            # [MODIFY] agregar zookeeper,
kafka, schema-registry services
├── src/main/resources/
│   ├── application.yml                           # [MODIFY] kafka health check
config
│   ├── application-local.yml                     # [MODIFY] kafka bootstrap-
servers, schema registry URL
│   └── kafka/schemas/
│       └── signature-event.avsc                  # [CREATE] Avro schema definition
├── src/main/java/com/bank/signature/infrastructure/config/
│   ├── KafkaConfig.java                          # [CREATE] KafkaTemplate,
ProducerFactory beans
│   └── KafkaTopicConfig.java                     # [CREATE] NewTopic beans
(signature.events, dlq)
├── src/test/java/com/bank/signature/infrastructure/
│   └── KafkaInfrastructureIntegrationTest.java  # [CREATE] @EmbeddedKafka test
├── target/generated-sources/avro/                # [AUTO-GENERATED] Avro classes
(mvn compile)
│   └── com/bank/signature/event/
│       └── SignatureEvent.java
├── docs/development/
│   └── kafka-messaging.md                        # [CREATE] Kafka documentation
└── README.md                                      # [MODIFY] Kafka setup section
```

Testing Standards Summary

- **Unit Tests**: No aplicable (Kafka configuration es infrastructure setup)
- **Integration Tests**:
  - `KafkaInfrastructureIntegrationTest.java` con `@EmbeddedKafka`
  - Verifica KafkaTemplate puede enviar GenericRecord
  - Verifica serialización Avro correcta
  - Verifica Schema Registry (mock) valida esquema
- **Manual Tests**:
  - `docker-compose up -d` levanta Kafka cluster
  - `curl http://localhost:8081/subjects` lista schemas
  - `/actuator/health/kafka` retorna UP
- **CI/CD Pipeline**:

    - Docker Compose up en pipeline
    - `mvn verify` (incluye Embedded Kafka tests)
    - Validar health check UP

Project Structure Notes

- **Avro Schema Evolution**: BACKWARD compatibility permite agregar campos opcionales sin romper consumers
- **Event Types**: 8 event types definidos (alineados con `docs/architecture/04-event-catalog.md`)
- **Partitioning**: `aggregateId` (signature_request.id) como partition key garantiza orden por request
- **DLQ (Dead Letter Queue)**: `signature.events.dlq` para mensajes fallidos (retry exhausted)
- **Schema Registry Subject**: TopicNameStrategy → `signature.events-value` (un schema por topic)

References

- **[Source: docs/architecture/04-event-catalog.md]**: Catálogo de 8 eventos de dominio
    - SIGNATURE_REQUEST_CREATED, CHALLENGE_SENT, CHALLENGE_COMPLETED, CHALLENGE_FAILED, SIGNATURE_COMPLETED, SIGNATURE_FAILED, FALLBACK_TRIGGERED, PROVIDER_DEGRADED
- **[Source: docs/sprint-artifacts/tech-spec-epic-1.md]**: Kafka technology stack
    - Kafka 3.6 (Confluent), Schema Registry 7.5, Avro serialization
- **[Source: docs/epics.md]**: Story 1.3 acceptance criteria
    - Topics: signature.events (12 partitions), signature.events.dlq
    - Producer: acks=all, compression=snappy
- **[Source: docs/prd.md]**: Event Publishing requirements (FR39-FR46)
    - Atomicidad (Outbox pattern - Story 1.2), serialización Avro, partitioning por aggregate_id

Critical Implementation Notes

- **Kafka Advertised Listeners**: Docker Compose debe configurar `KAFKA_ADVERTISED_LISTENERS` con `PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL://kafka:29092` para que app en host y containers puedan conectarse
- **Schema Registry Compatibility**: `BACKWARD` mode permite agregar campos opcionales,

eliminar campos con defaults

- **Idempotence**: `enable.idempotence=true` + `acks=all` garantiza exactly-once semantics (no duplicados en caso de retry)
- **Avro Maven Plugin**: Genera clases Java en `target/generated-sources/avro/`, debe agregarse a classpath (maven-compiler-plugin source path)
- **Embedded Kafka Test**: Puerto debe ser diferente (9093) para no colisionar con Kafka en Docker (9092)

Definition of Done

- [ ] **Code Complete**:
  - [ ] 3 servicios agregados a `docker-compose.yml` (zookeeper, kafka, schema-registry)
  - [ ] 5 dependencies agregadas a `pom.xml` (spring-kafka, kafka-avro-serializer, avro, test utils)
  - [ ] Avro schema `signature-event.avsc` definido con 8 event types
  - [ ] `KafkaConfig.java` y `KafkaTopicConfig.java` creados
  - [ ] `KafkaInfrastructureIntegrationTest.java` creado con `@EmbeddedKafka`
  - [ ] Maven Avro Plugin configurado en `pom.xml`
  - [ ] `application-local.yml` configurado con Kafka properties
- [ ] **Tests Passing**:
  - [ ] Integration test con `@EmbeddedKafka` pasa en `mvn verify`
  - [ ] Manual test: `docker-compose up -d` levanta Kafka cluster exitosamente
  - [ ] Manual test: `curl http://localhost:8081/subjects` lista schemas
  - [ ] Manual test: `/actuator/health/kafka` retorna UP
- [ ] **Architecture Validated**:
  - [ ] Avro schema sigue naming conventions (namespace: `com.bank.signature.event`)
  - [ ] Topics configurados con partitioning strategy (12 partitions para throughput)
  - [ ] Idempotent producer habilitado (`enable.idempotence=true`)
  - [ ] DLQ topic configurado para mensajes fallidos
- [ ] **Documentation Updated**:
  - [ ] `README.md` incluye sección "Kafka Setup" con comandos Docker Compose, verificación topics/schemas
  - [ ] `docs/development/kafka-messaging.md` creado con guidelines de schema evolution, testing, troubleshooting

    - [ ] `KafkaConfig.java` tiene comentarios explicando configuraciones (acks, idempotence, compression)
    - [ ] `CHANGELOG.md` actualizado: "Added Kafka 3.6 + Schema Registry 7.5 with Avro serialization"
- [ ] **Code Review Approved**:
    - [ ] Peer review confirma Avro schema es backward compatible
    - [ ] Verificar Kafka advertised listeners configurados correctamente (localhost + internal)
    - [ ] Validar idempotence + acks=all para exactly-once semantics
    - [ ] Confirmar topics tienen retention policies correctas (7 días events, 30 días DLQ)
- [ ] **Story Marked as Done**:
    - [ ] Todos los 12 Acceptance Criteria verificados ✅
    - [ ] Sprint status actualizado: `1-3-kafka-infrastructure-schema-registry: done`
    - [ ] Story list actualizada en `docs/sprint-artifacts/sprint-status.yaml`

Dev Agent Record

Context Reference

- `docs/sprint-artifacts/1-3-kafka-infrastructure-schema-registry.context.xml`

Agent Model Used

Claude Sonnet 4.5

Debug Log References

Completion Notes List

File List

Created:

Modified:

Deleted:

Change Log

Date	Author	Change
2025-11-26	BMAD SM Agent	Story 1.3 draft created with Kafka + Schema Registry + Avro
2025-11-26	BMAD SM Agent	Technical context generated, status: ready-for-dev